509, 876

(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

04 OCT 2004

(43) Date de la publication internationale
9 octobre 2003 (09.10.2003)

PCT

(10) Numéro de publication internationale
WO 03/083645 A2

(51) Classification internationale des brevets[7] : G06F 7/72

(21) Numéro de la demande internationale :
PCT/FR03/01058

(22) Date de dépôt international : 3 avril 2003 (03.04.2003)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
02/04117 3 avril 2002 (03.04.2002) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **GEMPLUS** [FR/FR]; Avenue du Pic de Bertagne, Parc d'Activités de Gemenos, F-13420 Gemenos (FR).

(72) Inventeurs; et
(75) Inventeurs/Déposants *(pour US seulement)* : **JOYE, Marc** [BE/FR]; 19, rue Voltaire, F-83640 Saint Zacharie (FR). **CHEVALLIER-MAMES, Benoît** [FR/FR]; Résidence Le Général, 14, boulevard Ganteaume, F-13400 Aubagne (FR).

(74) Mandataire : **BRUN, Philippe**; c/o Gemplus, Service brevets, La Vigie, PB 90, F-13705 La Ciotat Cedex (FR).

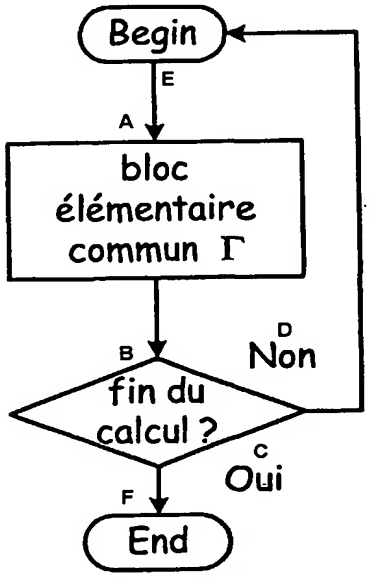
(81) États désignés *(national)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) États désignés *(régional)* : brevet ARIPO (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), brevet eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), brevet européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI,

[Suite sur la page suivante]

(54) Title: CRYPTOGRAPHIC METHOD PROTECTED AGAINST COVERT CHANNEL TYPE ATTACKS

(54) Titre : PROCEDE CRYPTOGRAPHIQUE PROTEGE CONTRE LES ATTAQUES DE TYPE A CANAL CACHE



A...BLOC ÉLÉMENTAIRE COMMUN:- COMMON ELEMENTARY BLOCK
B...FIN DE CALCUL?:- END OF CALCULATION?
C...OUI:- YES
D...NON:- NO
E...DEBUT
F...FIN

(57) **Abstract:** The invention relates to a cryptographic method secured against a covert channel attack. According to the invention, in order to carry out a selected block of instructions (?j) as a function of an input variable (D_1) amongst N predefined instruction blocks (?1,, ?<SB>N</SB>), a common block (Γ (k,s)) is carried out on the predefined N instruction blocks (?1,, ?<SB>N</SB>), a predefined number (L_j) of times, the predefined number (L_j) being associated with the selected instruction block (?j).

(57) **Abrégé :** L'invention concerne un procédé cryptographique sécurisé contre une attaque à canal caché. Selon l'invention, pour exécuter un bloc d'instructions choisi (Πj) en fonction d'une variable d'entrée (D_1) parmi N blocs d'instructions prédéfinis (Π_1, , Π_N), on exécute un nombre prédéfini (L_j) de fois un bloc commun ((k,s)) aux N blocs d'instructions prédéfinis (Π_1, , Π_N), le nombre prédéfini (L_j) étant associé au bloc d'instructions choisi (Π_j).

WO 03/083645 A2

FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), brevet OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Déclarations en vertu de la règle 4.17 :

— *relative à l'identité de l'inventeur (règle 4.17.i)) pour les désignations suivantes AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW, brevet ARIPO (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), brevet eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), brevet européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), brevet OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)*

— *relative au droit du déposant de demander et d'obtenir un brevet (règle 4.17.ii)) pour les désignations suivantes AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW, brevet ARIPO (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), brevet eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), brevet européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), brevet OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)*

— *relative au droit du déposant de revendiquer la priorité de la demande antérieure (règle 4.17.iii)) pour toutes les désignations*

— *relative à la qualité d'inventeur (règle 4.17.iv)) pour US seulement*

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

PROCEDE CRYPTOGRAPHIQUE PROTEGE CONTRE LES ATTAQUES DE TYPE A CANAL CACHE

L'invention concerne un procédé cryptographique protégé contre les attaques de type à canal caché. L'invention est notamment intéressante pour protéger des algorithmes au cours duquel on exécute un bloc d'instructions parmi plusieurs blocs d'instructions différents en fonction d'une variable d'entrée. Un tel algorithme est par exemple, mais non limitativement, un algorithme binaire d'exponentiation, réalisant un calcul de type $B = A^D$, A, B et D étant des nombres entiers. Un tel algorithme est par exemple mis en œuvre dans des dispositifs électroniques tels que des cartes à puce.

Le diagramme de principe d'un tel algorithme est représenté sur la figure 1. Il comprend une première étape de test de la valeur d'une donnée d'entrée. En fonction du résultat du test, on réalise ensuite un bloc d'instructions Π_0 ou un bloc d'instructions Π_1. L'algorithme peut ensuite se terminer, ou bien une nouvelle étape de test est réalisée sur une autre variable d'entrée. Dans l'exemple d'une opération de type $B = A^D$, la variable d'entrée est un bit D_i de D et le diagramme de la figure 1 est répété successivement pour chaque bit de D.

Les blocs d'instructions Π_0, Π_1 comprennent chacun un ensemble d'instructions à exécuter, par exemple des opérations d'addition, de multiplication, de mise à jour de variable, etc. Le nombre et /ou le type des instructions peuvent être différents d'un bloc d'instructions Π_0, Π_1 à l'autre.

De nombreux algorithmes cryptographiques sont basés sur le diagramme de principe de la figure 1. C'est notamment le cas des algorithmes cryptographiques basés

sur des calculs d'exponentiation du type $B = A^D$, où A, B sont des nombres entiers le plus souvent de grande taille, et D un nombre prédéterminé de M bits.

Les nombres A, B peuvent correspondre par exemple à un texte chiffré ou à chiffrer, une donnée signée ou à signer, une donnée vérifiée ou à vérifier, etc. Le nombre D peut correspondre à des éléments de clés, privées ou publiques utilisées pour le chiffrage ou le déchiffrage des nombres A, B.

A titre d'exemple des algorithmes tels que l'algorithme dit "Square-And-Multiply", l'algorithme dit "Right-To-Left binary algorithm", l'algorithme dit "(M, M^3) algorithm" peuvent être utilisés pour réaliser des calculs d'exponentiation.

Un utilisateur malveillant peut éventuellement engager des attaques, visant à découvrir notamment des informations confidentielles (comme par exemple la clé D ou une donnée dérivée de cette clé) manipulées dans des traitements effectués par le dispositif de calcul exécutant une opération d'exponentiation.

Une attaque simple, dite "timing attack", contre l'algorithme de la figure 1 consiste à mesurer le temps nécessaire au dispositif pour exécuter un bloc d'instructions entre deux étapes de test. Si les temps d'exécution des blocs d'instructions Π_0, Π_1 sont différents, alors il est facile d'identifier un bloc d'instructions Π_0 ou Π_1 et d'en déduire la valeur de la variable d'entrée associée.

Pour se prémunir contre cette attaque, il est possible d'ajouter des instructions fictives dans le bloc d'instructions Π_0 ou Π_1 le plus court (un bloc d'instruction est " le plus court" si le temps mis pour le réaliser est le plus petit) de sorte à ce que les deux blocs d'instructions Π_0, Π_1 soient de même durée.

considérée, selon la valeur de ce bit et / ou selon l'instruction.

Les attaques à canaux cachés peuvent aboutir sur des algorithmes tels que celui de la figure 1 si les blocs d'instructions Π_0, Π_1 ne sont pas équivalents vis-à-vis de ces attaques.

Le terme "équivalent" doit être compris ici et dans tout ce qui suit de la manière suivante. Deux instructions $INST_1$, $INST_2$ (ou deux blocs d'instructions Π_0, Π_1) sont dits *équivalentes* (on note $INST_0 \sim INST_1$) s'il n'est pas possible de les différencier par une attaque à canal caché. C'est le cas notamment si la grandeur physique mesurée au cours de l'attaque suit la même évolution pour les deux instructions. A noter toutefois que deux instructions peuvent être équivalentes vis-à-vis d'une attaque à canal caché et ne pas être équivalentes vis-à-vis d'une autre attaque à canal caché.

Dans le même esprit, on dira que deux instructions (ou blocs d'instructions) sont *égales* si, lorsqu'elles sont utilisées avec les mêmes données d'entrée, elles produisent des données de sortie identiques.

Il est connu de se prémunir contre les attaques à canaux cachés en ajoutant des instructions fictives à l'algorithme. On supposera dans ce qui suit qu'une instruction fictive est équivalente à une instruction réelle similaire. Par exemple l'instruction i<-i-0 est supposée équivalente à l'instruction i<-i-1.

Dans le cas de l'algorithme de la figure 1, il est ainsi connu de réaliser un bloc d'instructions Π_1 fictif après chaque bloc d'instructions Π_0, et de réaliser de manière symétrique un bloc d'instructions Π_0 fictif avant chaque bloc d'instructions Π_1 (voir les étapes en pointillés sur la figure 1). Ainsi, quelle que soit la valeur de la donnée d'entrée, on réalisera un bloc

d'instructions Π_0 et un bloc d'instructions Π_1, dans l'ordre, l'un ou l'autre étant fictif, de sorte qu'il ne soit pas possible de prédire la valeur de la donnée d'entrée, les grandeurs physiques relatives à un calcul étant équivalentes. On note ainsi :

$$(\Pi_0 \,||\, \Pi_{1(fictif)}) \sim (\Pi_{0(fictif)} \,||\, \Pi_1).$$

La notation "||" signifie la réalisation successive de blocs d'instructions Π_0, Π_1, (ou plus généralement la réalisation successive de deux instructions).

Si cette solution est efficace contre les attaques à canaux cachés, elle présente cependant l'inconvénient de multiplier en moyenne par deux le temps d'exécution de l'algorithme.

En effet, dans le cas d'un algorithme non protégé utilisant M données d'entrée (par exemple les M bits d'une donnée D), on réalise statistiquement en moyenne M/2 blocs d'instructions Π_0 et M/2 blocs d'instructions Π_1. Si T0, respectivement T1 sont les temps moyens d'exécution d'un bloc d'instructions Π_0, respectivement Π_1, alors le temps moyen d'exécution de l'algorithme non protégé est égal à M*(T0 + T1) / 2.

Par contre, dans le cas de l'algorithme protégé par des blocs d'instructions Π_0, Π_1 fictifs, on réalise systématiquement un bloc d'instructions Π_0 et un bloc d'instructions Π_1 pour chacune des M données d'entrée. En conséquence, le temps moyen d'exécution de l'algorithme protégé par des blocs d'instructions fictifs est égal à M*(T0 + T1).

Un premier but de l'invention est de proposer un nouvel algorithme cryptographique sécurisé contre les attaques à canaux cachés. Un deuxième but de l'invention est un procédé cryptographique protégé et plus rapide que les algorithmes protégés existants.

Ce but est atteint par un procédé de calcul cryptographique selon l'invention, caractérisé en ce que, pour exécuter un bloc d'instructions choisi (Π_j) en fonction d'une variable d'entrée (D_i) parmi N blocs d'instructions prédéfinis (Π_1, ..., Π_N), on exécute un nombre prédéfini (L_j) de fois un bloc commun ($\Gamma(k,s)$) aux N blocs d'instructions prédéfinis (Π_1, ..., Π_N), le nombre prédéfini (L_j) étant associé au bloc d'instructions choisi (Π_j).

En d'autres termes, selon l'invention, on réalise un seul bloc élémentaire, le bloc élémentaire commun, quelle que soit la variable d'entrée. Le bloc élémentaire commun est exécuté un nombre prédéfini de fois, selon la variable d'entrée. Contrairement aux procédés connus, on n'exécute pas des blocs d'instructions différents en fonction de la variable d'entrée.

Ainsi, avec l'invention, Il n'est alors pas possible de déterminer par une attaque à canal caché quel bloc d'instructions est exécuté. Un procédé selon l'invention est donc bien protégé.

Le nombre prédéfini (L_j) est variable d'un bloc d'instructions prédéfini (Π_1, ..., Π_N) à l'autre.

Le bloc commun ($\Gamma(k,s)$) comprend de préférence au moins une instruction de calcul ($\gamma(k,s)$) équivalente vis-à-vis d'une attaque à canal caché à une instruction de calcul de chaque bloc prédéfini (Π_1, ..., Π_N).

Le bloc commun ($\Gamma(k,s)$) peut également comprendre une instruction de mise à jour d'un pointeur de boucle (k) indiquant un nombre d'exécutions déjà exécutées du bloc commun ($\Gamma(k,s)$).

Si nécessaire, le bloc commun ($\Gamma(k,s)$) peut comprendre en complément une instruction de mise à jour d'un pointeur d'état (s) indiquant si le nombre prédéfini (L_j) a été atteint.

La valeur du pointeur de boucle (k) et / ou la valeur du pointeur d'état (s) sont fonction de la valeur

de la variable d'entrée (D_i) et / ou du nombre d'instructions du bloc d'instructions (Π_j) associé à la valeur de la variable d'entrée.

De préférence, si plusieurs blocs communs sont possibles, le bloc commun est choisi minimal, en ce sens qu'il comprend un nombre d'instructions minimal et / ou qu'il s'effectue en un temps minimal.

De préférence encore, pour réaliser successivement plusieurs blocs d'instructions choisis parmi les N blocs d'instructions prédéfinis (Π_1, ..., Π_N), chaque bloc d'instruction choisi étant sélectionné en fonction d'une variable d'entrée (D_i) associée à un indice d'entrée (i),

on exécute un nombre total (L_T) de fois le bloc commun ($\Gamma(k,s)$), le nombre total (L_T) étant égal à une somme des nombres prédéfinis (L_j) associés à chaque bloc d'instructions choisi (Π_j).

Là encore, pour exécuter successivement plusieurs blocs d'instructions, on exécute seulement le bloc commun un nombre approprié de fois ; ceci quels que soient les blocs d'instructions à exécuter. Il n'est donc pas possible de prédire à quel bloc d'instructions est associé le bloc commun en cours d'exécution. Une attaque à canal caché ne peut donc pas aboutir.

A noter qu'un même bloc d'instructions (Π_j) peut être choisi plusieurs fois selon la variable d'entrée associée (Di) à l'indice d'entrée (i).

Selon un mode de réalisation de l'invention, une ou plusieurs relations mathématiques sont utilisées pour mettre à jour le pointeur de boucle et / ou le pointeur d'état et / ou des indices de registres utilisés pour la mise en œuvre du procédé cryptographique et / ou la ou les variables d'entrées. Selon un autre mode de réalisation de l'invention, la mise à jour se fait en utilisant une table à plusieurs entrées. Ces modes de

réalisation seront détaillés plus longuement par la suite à l'aide d'exemples pratiques.

L'invention concerne également un procédé d'obtention d'un bloc commun ($\Gamma(k,s)$) à N blocs d'instructions (Π_1, ..., Π_N) prédéfinis. Ledit procédé est susceptible d'être utilisé pour la mise en œuvre d'un procédé de calcul cryptographique sécurisé selon l'invention et tel que celui décrit ci-dessus.

Selon l'invention, on obtient un bloc commun ($\Gamma(k,s)$) en réalisant les étapes suivantes :

E1 : décomposition de chaque bloc d'instructions prédéfini (Π_1, ..., Π_N) en une suite de blocs élémentaires (γ) équivalents vis-à-vis d'une attaque à canal caché, et classement de l'ensemble des blocs élémentaires (par exemple par attribution d'un rang),

E2 : recherche d'un bloc élémentaire commun ($\gamma(k,s)$) équivalent à tous les blocs élémentaires (γ) de tous les blocs d'instructions prédéfinis,

E3 : recherche d'un bloc commun ($\Gamma(k,s)$) comprenant au moins le bloc élémentaire commun ($\gamma(k,s)$) précédemment obtenu lors de l'étape E2 et une instruction de mise à jour d'un pointeur de boucle (k) telle que une exécution du bloc élémentaire commun associée à la valeur du pointeur de boucle (k) et une exécution du bloc élémentaire de rang égal à la valeur du pointeur de boucle (k) soient identiques.

Si nécessaire, au cours de l'étape E1, une ou des instructions fictives peuvent être ajoutées à la série des instructions d'un ou plusieurs blocs d'instructions. Ceci peut faciliter la décomposition de chaque blocs d'instructions en des blocs élémentaires tous équivalents vis-à-vis d'une attaque à canal caché.

Au cours de l'étape E1, on découpe chaque bloc d'instructions prédéfinis Π_1 à Π_N en blocs élémentaires

équivalents vis-à-vis d'une attaque donnée ; les blocs élémentaires sont classés. Par exemple :

$$\Pi_1 = \gamma 1 \,||\, \gamma 2 \,||\, \gamma 3 \;;\; \Pi_2 = \gamma 4 \,||\, \gamma 5 \;;\; \ldots$$

Plus généralement, chaque bloc d'instructions Π_1, ..., Π_N se décompose ainsi :

$$\Pi_1 = \gamma(C_1) \,||\, \ldots \,||\, \gamma(C_1+L_1-1),$$
$$\Pi_2 = \gamma(c_2) \,||\, \ldots \,||\, \gamma(c_2+L_2-1),$$
$$\ldots$$
$$\Pi_j = \gamma(c_j) \,||\, \ldots \,||\, \gamma(c_j+L_j-1),$$
$$\ldots$$
$$\Pi_N = \gamma(c_N) \,||\, \ldots \,||\, \gamma(c_N+L_N-1)$$

avec $C_1 = 0$

$C_2 = L_1$

...

$C_j = L_1+L_2+ \ldots + L_{j-1}$

...

$C_N = L_1+ \ldots L_{N-1}$

L_j est le nombre de blocs élémentaires nécessaires pour décomposer complètement le bloc prédéfini d'instructions Π_j.

Au cours de l'étape E2, on recherche un bloc élémentaire commun γ tel que chaque bloc d'instructions Π_j ($1 \leq j \leq N$) peut être exprimé sous forme d'une répétition L_j fois du bloc élémentaire commun γ.

De préférence, le bloc commun est choisi minimal. En d'autres termes, il comprend un nombre minimum d'instructions et / ou il s'exécute en un temps minimal.

Au cours de l'étape E3, on recherche un bloc commun comprenant :

- un ou plusieurs blocs élémentaires communs obtenus lors de l'étape E2, et
- une instruction de mise à jour d'un pointeur de boucle (k) tel que une exécution du bloc élémentaire commun associée à la valeur du pointeur de boucle (k) et une exécution du bloc élémentaire de rang égal à la valeur du pointeur de boucle (k) soient identiques.

Si nécessaire, un pointeur d'état s peut également être utilisé en complément du pointeur de boucle :

- le pointeur d'état s indique si le bloc élémentaire commun a déjà été exécuté un nombre prédéfini de fois correspondant au nombre L_j de blocs élémentaires décomposant un bloc d'instructions Π_j donné ; dans un exemple, le pointeur d'état s est égal à 1 lorsque le nombre prédéfini L_j de blocs élémentaires a été exécuté, et il est égal à 0 sinon.

- le pointeur de boucle indique le rang du bloc élémentaire à exécuter parmi l'ensemble des blocs élémentaires. De manière très générale, le pointeur de boucle peut être défini dans tous les cas selon l'équation 1 suivante :

$$k <- (/s).(k+1) + s.f(D_i)$$

D_i est la variable d'entrée permettant de sélectionner un bloc d'instructions à exécuter, s est le pointeur d'état, et f est une fonction logique de la variable d'entrée D_i associée à un bloc d'instructions prédéfini Π_j à exécuter, et /s est le complément du pointeur s (fonction NON logique).

L'équation ci-dessus donnant la valeur de k est obtenue par le raisonnement suivant.

Lorsqu'on réalise un bloc d'instructions Π_j, le pointeur de boucle k doit être incrémenté de 1 à chaque exécution du bloc élémentaire commun (associée à un bloc élémentaire équivalent de la décomposition du bloc Π_j) tant que s = 0, c'est-à-dire tant que le nombre de blocs élémentaires associés au bloc Π_j n'a pas été atteint. Ceci se traduit par l'instruction :

$$k <- (k+1) \text{ lorsque } s = 0$$

Inversement, lorsqu'on réalise le bloc élémentaire commun associé au dernier bloc élémentaire du bloc Π_j (c'est-à-dire lorsque s = 1), il est nécessaire de modifier k de sorte à réaliser le bloc élémentaire commun

associé au premier bloc élémentaire du bloc d'instructions suivant $\Pi_{j'}$. Ceci se traduit par l'instruction suivante :

$k <- f(D_i)$ lorsque $s = 1$

où D_i est la variable d'entrée qui conditionne le choix du calcul $\Pi_{j'}$ à réaliser.

En combinant les deux dernières instructions, on obtient finalement l'équation 1.

L'équation ci-dessus donnant la valeur de k en fonction de s est valable dans tous les cas. Dans certains cas particuliers, cette équation peut être modifiée comme on le verra mieux plus loin dans des exemples pratiques.

L'invention et les avantages qui en découlent apparaîtront plus clairement à la lecture de la description qui suit d'exemples de mise en œuvre d'un procédé cryptographique sécurisé selon l'invention. La description est à lire en référence aux dessins annexés dans lesquels :

- la figure 1 est un diagramme générique de procédés connus et susceptibles d'être protégés selon l'invention,
- la figure 2 est un diagramme du procédé générique de la figure 1 protégé selon l'invention,
- les figures 3 et 4 détaillent la réalisation de certaines étapes du procédé de la figure 2 dans le cas de procédés d'exponentiation connus, protégés selon l'invention.

Dans les exemples qui suivent, on va décrire notamment l'obtention d'un bloc élémentaire commun selon l'invention et l'utilisation de ce bloc élémentaire, dans les cas pratiques de procédés de calcul cryptographique.

Exemple 1

Dans un premier exemple pratique, on considère un algorithme d'exponentiation de type "Square-and-Multiply", qui permet de réaliser une opération d'exponentiation de type $B = A^D$, où $D = (D_{M-1}, \ldots, D_0)$ est un nombre de M bits. La forme connue de cet algorithme peut être représentée comme suit :

Initialisation :
$R_0 <- 1$; $R_1 <- A$; $i <- M-1$
Tant que $i \geq 0$, répéter :
Si $D_i = 1$, alors réaliser Π_0 :
$R_0 <- R_0 \times R_0$
$R_0 <- R_0 \times R_1$
$i <- i-1$
Si $D_i = 0$, alors réaliser Π_1 :
$R_0 <- R_0 \times R_0$
$i <- i-1$
Retourner R_0.

Algorithme 1 "Square-and-Multiply" non-protégé

R_0, R_1 sont des registres d'un dispositif de calcul adapté pour la mise en œuvre de l'algorithme, et i est un indice de boucle repérant les différents bits de D. Selon la valeur de D_i, on exécute $\Pi_j = \Pi_0$ ou $\Pi_j = \Pi_1$.

Dans l'algorithme 1, les blocs d'instructions Π_0, Π_1 sont réalisés selon la valeur d'un bit D_i de l'exposant D, et on décrémente l'indice de boucle i à la fin de chaque bloc d'instructions Π_0, Π_1 de sorte à traiter successivement tous les bits D_i de D.

Dans l'algorithme 1, les bloc d'instructions Π_0, Π_1 ne sont pas équivalents vis-à-vis d'une attaque à canal caché, notamment parce ce que le nombre d'instructions de Π_0 est différent du nombre d'instructions de Π_1.

Pour protéger l'algorithme 1 selon l'invention, on recherche un bloc élémentaire commun Γ susceptible d'être utilisé pour l'exécution des blocs Π_0, Π_1.

Pour cela, on décompose tout d'abord chaque bloc d'instructions Π_0, Π_1 en une suite de blocs élémentaires, tous équivalents entre eux vis-à-vis d'une attaque donnée.

Le bloc d'instructions Π_0 peut s'écrire :

$R_0 <- R_0 \times R_0$
i <- i-0
$R_0 <- R_0 \times R_1$
i <- i-1

L'instruction i <- i-0 est fictive : elle ne modifie aucune variable, aucune donnée manipulée par l'algorithme 1.

Π_0 peut alors se décomposer en deux blocs élémentaires : $\Pi_0 = \gamma 0 \,||\, \gamma 1$ avec

$\gamma 0$: $R_0 <- R_0 \times R_0$
i <- i-0
$\gamma 1$: $R_0 <- R_0 \times R_1$
i <- i-1

Π_1 se décompose de la même façon en un bloc élémentaire :

$\Pi_1 = \gamma 2$ avec

$\gamma 2$: $R_0 <- R_0 \times R_0$
i <- i-1

On note que les blocs $\gamma 0$, $\gamma 1$, $\gamma 2$ sont tous équivalents ($\gamma 0 \sim \gamma 1 \sim \gamma 2$) vis à vis d'une attaque à canal caché si on suppose que les instructions $R_0 <- R_0 \times R_0$ et $R_0 <- R_0 \times R_1$ sont équivalentes et que les instructions i <- i-0 et i <- i-1 sont équivalentes.

On a ainsi décomposé chaque bloc d'instructions Π_0, Π_1 en un nombre variable (d'un bloc d'instructions à l'autre) de blocs élémentaires, tous équivalents les uns aux autres.

On définit ensuite un pointeur d'état s et un pointeur de rang k. Lorsqu'un bloc d'instructions Π_j est en cours d'exécution :

- k est utilisé pour indiquer quel bloc élémentaire γk doit être réalisé ; la valeur de k dépend notamment du bloc Π_j en cours d'exécution (et donc de la variable d'entrée D_i testée) et de l'état d'avancement de l'exécution du bloc Π_j
- s est utilisé pour indiquer si au moins un bloc élémentaire γk doit être encore être réalisé ou si le bloc γk en cours est le dernier du bloc d'instructions Π_j.

Dans le cas de l'exemple ci-dessus relatif à l'algorithme 1, on peut résumer l'évolution des pointeurs k, s par le tableau ci-dessous.

		k	s
(D_i = 1)	$\gamma 0$: R_0 <- $R_0 \times R_0$; i <- i-0	0	0
	$\gamma 1$: R_0 <- $R_0 \times R_1$; i <- i-1	1	1
(D_i = 0)	$\gamma 2$: R_0 <- $R_0 \times R_0$; i <- i-1	2	1

tableau 1

s peut être calculé à partir de k : si le bloc élémentaire γk qui doit être réalisé est le dernier bloc élémentaire d'un bloc Π, alors s = 1, sinon s = 0.

Dans le cas de l'algorithme 1, on peut par exemple calculer s par la relation suivante :

$$s = (k \bmod 2) + (k \operatorname{div} 2) \qquad \text{(EQ a)}$$

"div" désigne une division entière et "mod" une réduction modulaire. A partir de l'équation 1, on retrouve les différentes valeurs de s en fonction de k (cf tableau 1).

La mise à jour de k peut être obtenue à partir de s et de D_i, D_i étant représentatif du bloc Π_j en cours :

- si s = 0 (bloc Π_j en cours de réalisation), k est incrémenté de 1 à chaque réalisation d'un bloc

élémentaire γ , pour effectuer ensuite le bloc élémentaire γ suivant.

- si s = 1, le bloc Π en cours est terminée et le bloc élémentaire γ suivant à effectuer est le premier bloc élémentaire du prochain bloc Π_j à exécuter ; k dépend donc de D_i.

De ce qui précède, on en déduit que k peut être obtenu par la relation suivante :

$$k \leftarrow (/s)\times(k+1) + s\times f(D_i) \qquad \text{(EQ b)}$$

/s est la valeur complémentaire de s (fonction NON logique), et f est une fonction logique de D_i, qui dépend de l'algorithme à protéger (voir aussi figure 3).

Dans le cas de l'algorithme 1, on peut par exemple choisir $f(D_i) = 2\times(/D_i)$

Ainsi, avec l'équation 3 :

$$k \leftarrow (/s)\times(k+1) + s\times 2\times(/D_i), \qquad \text{(EQ c)}$$

on retrouve les différentes valeurs de k en fonction de s et de D_i (cf tableau 1).

On définit finalement un bloc élémentaire *commun* $\gamma(k, s)$, équivalent aux blocs élémentaires $\gamma 0$, $\gamma 1$, $\gamma 2$ et tel que $\gamma(0, 0) = \gamma 0$, $\gamma(1, 1) = \gamma 1$ et $\gamma(2, 1) = \gamma 2$.

Pour l'algorithme 1, on peut par exemple choisir :

$\gamma(k, s)$: $R_0 \leftarrow R_0 \times R_{k \bmod 2}$

$i \leftarrow i - s$

En utilisant le bloc élémentaire commun $\gamma(k, s)$, l'algorithme 1 peut finalement s'écrire (voir aussi figure 3) :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $i \leftarrow M-1$

Tant que $i \geq 0$, répéter le bloc commun $\Gamma(k, s)$:

$k \leftarrow (/s)\times(k+1) + s\times 2\times(/D_i)$

$s \leftarrow (k \bmod 2) + (k \operatorname{div} 2)$

$\gamma(k,s)$: $R_0 <- R_0 \times R_{k \bmod 2}$

$i <- i - s$

Retourner R_0.

Algorithme 1 ptotégé

(algorithme "Square-And-Multiply" protégé)

Dans cet algorithme, un seul bloc commun $\Gamma(k, s)$ est utilisé, quelles que soient les valeurs de D_i. Dit autrement, quelle que soit la valeur de D_i, on exécute la même instruction ou le même bloc d'instructions. Dans le cas où $D_i = 0$, on exécute une seule fois le bloc $\Gamma(k, s)$. Dans le cas où $D_i = 1$, on exécute successivement deux fois le bloc commun $\Gamma(k, s)$.

Quelles que soient les valeurs des pointeurs k, s et quelle que soit la valeur de D_i, le bloc $\Gamma(k, s)$ associé est équivalent, vis-à-vis d'une attaque à canal caché, au bloc $\Gamma(k, s)$ précédemment exécuté et au bloc $\Gamma(k, s)$ exécuté ensuite. En conséquence, il n'est pas possible de les distinguer les uns des autres, et il n'est pas possible de savoir à quel bloc d'instructions Π_j correspond le bloc commun $\Gamma(k, s)$ en cours d'exécution.

A noter, que par rapport à l'algorithme 1 non protégé, l'algorithme 1 protégé selon l'invention utilise le même nombre d'instructions de calcul (telles que des instructions de multiplication par exemple) pour aboutir au même résultat final. L'algorithme 1 protégé selon l'invention comprend simplement des étapes supplémentaires de mise à jour de pointeurs : de telles étapes sont beaucoup plus rapides et beaucoup moins consommatrices de ressources qu'une instruction de calcul telle qu'une multiplication. En conséquence, le temps d'exécution de l'algorithme protégé est quasiment le même que celui de l'algorithme 1 non protégé : Tex = 1.5*M*T0, T0 étant le temps d'exécution d'une multiplication.

A noter également que le bloc commun $\Gamma(k, s)$ n'est pas unique pour un même algorithme, comme on va le voir avec l'exemple 2.

Exemple 2

Dans le cas de l'algorithme "Square and Multiply", d'autres décompositions du bloc d'instructions Π_0 peuvent être envisagées, par exemple :

$\Pi_0 = \gamma'0 \,||\, \gamma'1$ avec

$\gamma'0$: $R_0 \leftarrow R_0 \times R_0$

$i \leftarrow i-1$

$\gamma'1$: $R_0 \leftarrow R_0 \times R_1$

$i \leftarrow i-0$

Cette décomposition est envisageable dans la mesure où l'instruction fictive $i \leftarrow i-0$ peut être exécutée à tout instant au cours du bloc Π_0. On constate dès lors que les blocs élémentaires $\gamma'0$ et $\gamma 2$ sont identiques. Le tableau 1 est alors modifié alors de la manière suivante.

		k	s
$(D_i = 1)$	$\gamma'0 : R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-1$	0	0
	$\gamma'1 : R_0 \leftarrow R_0 \times R_1$; $i \leftarrow i-0$	1	1
$(D_i = 0)$	$\gamma'0 : R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-1$	0	1

tableau 2

Le pointeur s devient ici superflu dans la mesure où seules deux blocs élémentaires sont possibles $\gamma'0$ et $\gamma'1$. On obtient finalement le bloc élémentaire commun $\gamma'(k, s)$ et l'algorithme protégé suivant (voir aussi figure 4) :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $i \leftarrow M-1$; $k \leftarrow 1$

Tant que $i \geq 0$, répéter le bloc commun $\Gamma'(k,s)$:

$k \leftarrow (D_i)$ ET $(/k)$

$\gamma'(s, k) : R_0 \leftarrow R_0 \times R_k$

i <- i - (/k)

Retourner R_0.

Algorithme 2 protégé

(algorithme "Square-And-Multiply" protégé, 2ème version)

Exemple 3

L'algorithme d'exponentiation dit "Right-To-Left binary algorithm" est assez similaire à l'algorithme "Square-And Multiply". Il permet de réaliser une opération de type $B = A^D$ en partant du bit le plus faible de D de la manière suivante :

Initialisation :

R_0 <- 1 ; R_1 <- A ; i <- 0

Tant que $i \leq M-1$, répéter :

Si $D_i = 1$, alors réaliser le bloc Π_0 :

R_0 <- $R_0 \times R_1$

R_1 <- $R_1 \times R_1$

i <- i+1

Si $D_i = 0$, alors réaliser le bloc Π_1 :

R_1 <- $R_1 \times R_1$

i <- i+1

Retourner R_0.

Algorithme dit "Right-To-Left binary algorithm"

Les blocs Π_0, Π_1 dans cet exemple peuvent se décomposer de la manière suivante :

	k	s
Π_0 ($D_i = 1$) $\gamma 0$: R_0 <- $R_0 \times R_1$; i <- i+0	0	0
$\gamma 1$: R_1 <- $R_1 \times R_1$; i <- i+1	1	1
Π_1 ($D_i = 0$) $\gamma 0$: R_1 <- $R_1 \times R_1$; i <- i+1	2	1

tableau 3

Ici également, comme seuls deux blocs élémentaires $\gamma 0$, $\gamma 1$ sont utilisés pour décomposer Π_0, Π_1, le pointeur s

est inutile. On peut par exemple choisir le bloc élémentaire commun $\gamma(k)$ suivant :

$\gamma(k)$: R_k <- $R_k \times R_1$
i <- $i+k$

et mettre à jour le pointeur k avant chaque réalisation du bloc $\gamma(k)$ en utilisant l'instruction k <- $k \oplus D_i$, où $\oplus$ désigne l'opérateur OU-Exclusif ($\oplus$). On obtient finalement l'algorithme 3 protégé ci-dessous :

Initialisation :
R_0 <- 1 ; R_1 <- A ; i <- 0 ; k <- 1
Tant que $i \leq M-1$, répéter le bloc $\Gamma(k, s)$:
k <- $k \oplus D_i$
$\gamma(k)$: R_k <- $R_k \times R_1$
i <- i+k
Retourner R_0.

Algorithme 3
("Right-To-Left binary algorithm" protégé)

Les exemples ci-dessus décrivent des algorithmes au cours desquels on exécute seulement deux blocs d'instructions Π_0 ou Π_1 en fonction de la valeur d'une variable d'entrée D_i. L'invention peut cependant s'appliquer à des algorithmes utilisant plus de deux blocs d'instructions Π.

Exemple 4

On considère dans cet exemple l'algorithme dit "(M, M^3) algorithm", connu sous la forme suivante :

Initialisation :
R_0 <- 1 ; R_1 <- A ; R_2 <- A^3 ;
D_{-1} <- 0 ; i <- M-1
Tant que $i \geq 0$, répéter :
Si $D_i = 0$, réaliser Π_0 :
R_0 <- $(R_0)^2$

$i \leftarrow i-1$

Si $D_i = 1$ ET $(D_{i-1} = 0)$, réaliser Π_1 :

$R_0 \leftarrow (R_0)^2$

$R_0 \leftarrow R_0 \times R_1$

$i \leftarrow i-1$

Si $D_i = 1$ ET $(D_{i-1} = 1)$, réaliser Π_2 :

$R_0 \leftarrow (R_0)^2$

$R_0 \leftarrow (R_0)^2$

$R_0 \leftarrow R_0 \times R_2$

$i \leftarrow i-2$

Retourner R_0.

Algorithme dit " (M, M^3) algorithm"

ET est la fonction ET logique. R_0, R_1, R_2 sont des registres du dispositif utilisé pour la mise en œuvre de l'algorithme.

En remplaçant les carrés de type $(R_0)^2$ par des multiplications de type $R_0 \times R_0$, et en introduisant des instructions fictives de type i <- i-0, on peut décomposer l'algorithme (M, M^3) selon le tableau :

		k	s
Π_0 $(D_i = 0)$	$\gamma 0$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-1$	0	1
Π_1 $(D_i = 1)$ et $(D_{i-1} = 0)$	$\gamma 1$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	1	0
	$\gamma 2$: $R_0 \leftarrow R_0 \times R_1$; $i \leftarrow i-1$	2	1
Π_2 $(D_i = 1)$ et $(D_{i-1} = 1)$	$\gamma 3$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	3	0
	$\gamma 4$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	4	0
	$\gamma 5$: $R_0 \leftarrow R_0 \times R_2$; $i \leftarrow i-2$	5	1

tableau 4

Le tableau 4 permet de calculer assez aisément la valeur du pointeur k en fonction de s et de D_i, et ainsi que la valeur du pointeur s en fonction de k, comme précédemment. Par ailleurs, les blocs $\gamma 0$ à $\gamma 5$ sont tous équivalents vis-à-vis d'une attaque à canal caché, et on

peut par exemple choisir le bloc élémentaire commun $\gamma(k, s)$ suivant :

$\gamma(k, s)$: $R_0 \leftarrow R_0 \times R_{sx(k \text{ div } 2)}$

$i \leftarrow i - sx(k \bmod 2 + 1)$

On en déduit finalement un algorithme 4 protégé :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $R_2 \leftarrow A^3$;

$D_{-1} \leftarrow 0$; $i \leftarrow M-1$; $s \leftarrow 1$

Tant que $i \geq 0$, répéter le bloc $\Gamma(k, s)$:

$k \leftarrow (/s) \times (k+1) + sx(D_i + 2 \times (D_i \text{ ET } D_{i-1}))$

$s \leftarrow /((k \bmod 2) \oplus (k \text{ div } 4))$

$\gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{sx(k \text{ div } 2)}$

$i \leftarrow i - sx(k \bmod 2 + 1)$

Retourner R_0.

Algorithme 4

(algorithme (M, M^3) protégé, 1ère version)

Exemple 5

Comme on l'a vu dans le cadre des exemples 1 et 2, pour un même algorithme, il est possible de choisir entre plusieurs blocs élémentaires communs $\gamma(k)$ ou $\gamma(k, s)$.

Dans le cas de l'algorithme (M, M^3) par exemple, il est également possible de décomposer les blocs Π_0, Π_1, Π_2 de la manière suivante :

		k	s
Π_0 ($D_i = 0$)	$\gamma 0$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-1$	0	1
Π_1 ($D_i = 1$) et ($D_{i-1} = 0$)	$\gamma 1$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	0	0
	$\gamma 2$: $R_0 \leftarrow R_0 \times R_1$; $i \leftarrow i-1$	1	1
Π_2 ($D_i = 1$) et ($D_{i-1} = 1$)	$\gamma 3$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	0	0
	$\gamma 4$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	1	0
	$\gamma 5$: $R_0 \leftarrow R_0 \times R_2$; $i \leftarrow i-2$	2	1

tableau 5

Par rapport au tableau 4, seules les valeurs de k ont été modifiées.

Le tableau 5 permet de calculer, comme précédemment, la valeur du pointeur k en fonction de s et de D_i, la valeur du pointeur s en fonction de k, et la valeur dont doit être décrémenté l'indice i. Par ailleurs, on peut par exemple choisir le bloc élémentaire commun $\gamma(k, s)$ suivant :

$\gamma(k, s)$: $R_0 \leftarrow R_0 \times R_{k \times s}$
$i \leftarrow i - k \times s - (/Di)$

On en déduit finalement un algorithme 5 protégé :

Initialisation :
$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $R_2 \leftarrow A^3$;
$D_{-1} \leftarrow 0$; $i \leftarrow M-1$; $s \leftarrow 1$
Tant que $i \geq 0$, répéter :
$k \leftarrow (/s) \times (k+1)$
$s \leftarrow s \oplus D_i \oplus ((D_{i-1}$ ET $(k \bmod 2))$
$\Gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{k \times s}$
$i \leftarrow i - k \times s - (/D_i)$
Retourner R_0.

Algorithme 5
(algorithme (M, M^3) protégé, $2^{ème}$ version)

Comme on a pu le voir dans les exemples précédents, il est assez simple d'obtenir, dans le cadre de l'invention, une décomposition de chaque bloc Π_j d'instructions en blocs élémentaires γ_0, γ_1, ..., γ_{Lj}.

Cependant, les relations liant le pointeur de boucle k, le pointeur d'état s à la variable D_i et / ou à la variable j indexant les différents blocs Π_0, Π_j, ..., Π_N deviennent complexes lorsque l'algorithme qu'on cherche à protéger est lui-même complexe (c'est-à-dire lorsqu'il utilise un nombre important de blocs Π_j différents, lorsque chaque bloc Π_j se décompose en un nombre important de blocs élémentaires γ, etc.). Pour certains algorithmes particulièrement complexes tels que les

algorithmes cryptographiques sur courbe elliptique, cette difficulté peut même s'avérer lourde voire insurmontable.

Pour résoudre ou contourner cette difficulté, selon un autre mode de réalisation de l'invention, les liens entre les valeurs du pointeur de boucle k, du pointeur d'état s, de l'indexe des registres utilisés, de l'indexe i de la variable D, de l'indexe j des blocs Π_j, etc. sont exprimés sous la forme d'une table U à plusieurs entrées comme on va le voir dans les exemples ci-dessous.

Dans la mise en œuvre pratique de l'invention, la dite table U pourra être par exemple mémorisée dans une mémoire, effaçable ou non, du dispositif utilisé. La mise à jour des pointeurs se fera alors par une lecture dans la mémoire d'une ou plusieurs valeurs dans la matrice U.

Exemple 6

On considère à nouveau la décomposition en blocs élémentaires de l'algorithme "Square and Multiply" :

		k	s
(D_i = 1)	$\gamma 0$: R_0 <- $R_0 \times R_0$; i <- i-0	0	0
	$\gamma 1$: R_0 <- $R_0 \times R_1$; i <- i-1	1	1
(D_i = 0)	$\gamma 2$: R_0 <- $R_0 \times R_0$; i <- i-1	2	1

tableau 6 = tableau 2

A chaque ligne du tableau 6 correspond une valeur différente de k. On peut écrire chaque bloc élémentaire γ_k sous la forme suivante :

$\gamma_k = [R_{U(k,0)} <- R_{U(k,1)} \times R_{U(k,2)} ; i <- i - U(k,3)]$

où U(k,1) est l'élément de la ligne k et de la colonne 1 de la matrice suivante :

$$(U_{k,l})_{\substack{0 \le k \le 2 \\ 0 \le l \le 3}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La matrice U est construite de la manière suivante. Chaque ligne de la matrice correspond à un bloc élémentaire γk d'indice k. A chaque colonne est associé un indice susceptible de varier d'un bloc élémentaire γk à l'autre. Ici, la colonne 0 est associée à l'indice du registre dans lequel est mémorisé le résultat de l'instruction R_α <- $R_\alpha \times R_\beta$ (α, β sont égaux à 0 ou 1 ici). La colonne 1 et la colonne 2 sont associées aux indices des registres dont on effectue le produit par l'instruction R_α <- $R_\alpha \times R_\beta$. Enfin, la colonne 3 est associée aux variations de l'index i. La matrice U s'obtient ainsi très simplement à partir du tableau récapitulant la décomposition des blocs Π_j en blocs élémentaires γk.

Les colonnes constantes de la matrice étant sans intérêt, elles peuvent être supprimées pour donner une matrice réduite, plus facile à mémoriser et à utiliser. On obtient ainsi le bloc élémentaire commun $\gamma(k)$:

$$\gamma(k) = [R_0 \text{ <- } R_0 \times R_{U(k,0)} \ ; \ i \text{ <- } i - U(k,1)]$$

avec, pour $0 \leq k \leq 2$ et $0 \leq l \leq 1$:

$$\left(U(k,l)\right)_{\substack{0 \leq k \leq 2 \\ 0 \leq l \leq 1}} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$$

On en déduit finalement l'algorithme complet protégé selon l'invention

Initialisation :

R_0 <- 1 ; R_1 <- A ; i <- M-1 ; s <- 1

Tant que $i \geq 0$, répéter le bloc $\Gamma(k, s)$:

k <- (/s)x(k+1) + sx2x(/D_i)

s <- U(k,1)

$\gamma(k, s)$: R_0 <- $R_0 \times R_{U(k,0)}$

i <- i - s

Retourner R_0

Algorithme 6

("Square and Multiply" protégé, 3ème version)

L'utilisation d'une matrice est une méthode très générale, beaucoup plus générale que les relations empiriques utilisées dans les exemples 1 à 5 pour expliciter les liens entre les différents indices utilisés.

L'expression des liens entre les indices sous la forme d'une matrice à plusieurs entrées présente l'avantage d'être beaucoup plus simple à mettre en œuvre et surtout d'être exploitable pour tous les algorithmes cryptographiques connus, y compris les plus complexes, comme on va le voir dans quelques exemples d'algorithmes de calcul cryptographique sur courbes elliptiques (exemples 8 et 9).

Exemple 7

On considère à nouveau ici l'algorithme (M, M3) et son tableau de décomposition :

		k	s
Π_0 $(D_i = 0)$	$\gamma 0$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-1$	0	1
Π_1 $(D_i = 1)$ et $(D_{i-1} = 0)$	$\gamma 1$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	1	0
	$\gamma 2$: $R_0 \leftarrow R_0 \times R_1$; $i \leftarrow i-1$	2	1
Π_2 $(D_i = 1)$ et $(D_{i-1} = 1)$	$\gamma 3$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	3	0
	$\gamma 4$: $R_0 \leftarrow R_0 \times R_0$; $i \leftarrow i-0$	4	0
	$\gamma 5$: $R_0 \leftarrow R_0 \times R_2$; $i \leftarrow i-2$	5	1

tableau 7 = tableau 4

Du tableau 7, on déduit aisément la matrice suivante :

$$(U(k,l))_{\substack{0\le k\le 5\\ 0\le l\le 2}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 2 & 1 \end{pmatrix}$$

une expression possible d'un bloc élémentaire commun $\gamma(k)$:

$$\gamma(k) = [R_0 <- R_0 \times R_{U(k,0)} \; ; \; i <- i - R_{U(k,1)}]$$

et un algorithme protégé utilisant le bloc élémentaire commun $\gamma(k)$:

Initialisation :

$R_0 <- 1$; $R_1 <- A$; $R_2 <- A^3$;

$i <- M-1$; $s <- 1$

Tant que $i \geq 0$, répéter le bloc commun $\Gamma(k, s)$:

$k <- (/s) \times (k+1) + s \times (D_i + 2 \times (/D_i \text{ ET } D_{i-1}))$;

$s <- U(k,2)$

$\gamma(k,s)$: $R_0 <- R_0 \times R_{U(k,0)}$;

$i <- i - U(k,1)$

Retourner R_0.

Algorithme 7

(algorithme (M, M^3) protégé, 3ème version)

Exemple 8

On considère un algorithme de calcul cryptographique sur une courbe elliptique E non super singulière définie sur un corps binaire $\mathcal{F}_{2q}$ par l'équation de Weierstrass suivante :

$$E/\mathcal{F}_{2q} : Y^2 + X \times Y = X^3 + a \times X^2 + b \qquad \text{(EQ d)}$$

où X, Y sont les coordonnées affines d'un point P de la courbe E.

Les opérations de base d'un algorithme cryptographique sur courbes elliptiques sont les opérations de doublement de points et les opérations d'addition de deux points distincts.

L'opération de doublement d'un point est définie par :

$$P3(X3, Y3) = 2 \times P1(X1, Y1) \text{ avec}$$

$$X3 = a + \lambda^2 + \lambda$$

$$Y3 = (X1+X3)\times\lambda + X3 + Y1$$
$$\text{et } \lambda = X1 + (Y1/X1)$$

L'opération d'addition de deux points distincts est définie par :

$$P(X3, Y3) = P1(X1, Y1) + P2(X2, Y2)$$
$$X3 = a + \lambda^2 + \lambda + X1 + X2$$
$$Y3 = (X1 + X3)\times\lambda + X3 + Y1$$
$$\text{et } \lambda = (Y1 + Y2) / (X1 + X2)$$

Dans le tableau 8, on a décomposé l'opération de doublement de points et l'opération d'addition de deux points distincts sous la forme chacune d'un bloc élémentaire $\gamma 0$, $\gamma 1$ équivalents (les mêmes opérations sont utilisées, éventuellement sur des registres différents) :

	k	s
$\gamma 0$: $R_1 \leftarrow R_1 + R_3$; $R_2 \leftarrow R_2 + R_4$; $R_5 \leftarrow R_2 / R_1$;	0	1
$R_1 \leftarrow R_1 + R_5$; $R_6 \leftarrow R_5^{\,2}$;		
$R_6 \leftarrow R_6 + a$; $R_1 \leftarrow R_1 + R_6$; $R_2 \leftarrow R_1 + R4$;		
$R_6 \leftarrow R_1 + R_3$; $R_5 \leftarrow R_5 \times R6$; $R_2 \leftarrow R_2 + R_5$		
$\gamma 1$: $R_6 \leftarrow R_1 + R_3$; $R_6 \leftarrow R_6 + R_3$; $R_5 \leftarrow R_2 / R_1$;	1	1
$R_5 \leftarrow R_1 + R_5$; $R_1 \leftarrow R_5^{\,2}$;		
$R_1 \leftarrow R_1 + a$; $R_1 \leftarrow R_1 + R_5$; $R_2 \leftarrow R_1 + R_2$;		
$R_6 \leftarrow R_1 + R_6$; $R_5 \leftarrow R_5 \times R_6$; $R_2 \leftarrow R_2 + R_5$		

tableau 8

Du tableau 8, on déduit la matrice suivante :

$$(U(k,l))_{\substack{0\le k\le 1\\ 0\le l\le 7}} = \begin{pmatrix} 1 & 2 & 4 & 1 & 6 & 6 & 4 & 3 \\ 6 & 6 & 3 & 5 & 1 & 5 & 2 & 6 \end{pmatrix}$$

La matrice comprend seulement deux lignes puisque seuls deux blocs élémentaires différents sont utilisés. La matrice comprend 8 colonnes, chacune associée à un indice de registre variant d'une ligne à l'autre. La colonne 0 est ainsi associée à l'indice du registre (R1 ou R6) dans lequel on mémorise le résultat de la 1ère

opération (R1 + R3), la colonne 1 est associée à l'indice du registre (R2 ou R6) dans lequel on mémorise le résultat de la 2ème opération ($R_2 + R_4$ ou $R_6 + R_3$), les colonnes 1 et 2 sont associées aux registres dont on additionne le contenu au cours de la 2ème opération ($R_2 + R_4$ ou $R_6 + R_3$), etc.

La matrice est à utiliser avec le bloc élémentaire commun suivant :

$\gamma(k)$: $R_U(k,0) \leftarrow R_1 + R_3$; $R_U(k,1) \leftarrow R_U(k,1) + R_U(k,2)$;
$R_5 \leftarrow R_2 / R_1$; $R_U(k,3) \leftarrow R_1 + R_5$;
$R_U(k,4) \leftarrow R_5^2$;
$R_U(k,4) \leftarrow R_U(k,4) + a$; $R_1 \leftarrow R_1 + R_U(k,5)$;
$R_2 \leftarrow R_1 + R_U(k,6)$; $R_6 \leftarrow R_1 + R_U(k,7)$;
$R_5 \leftarrow R_5 . R_6$; $R_2 \leftarrow R_2 + R_5$

pour réaliser un algorithme protégé utilisant le bloc commun $\Gamma(k)$ dans une boucle de type "répéter tant que" et réalisant une opération complexe utilisant des opérations de base (doublement de points et / ou addition de points)

Initialisation
$R_1 \leftarrow X_1$; $R_2 \leftarrow Y_1$;
$R_3 \leftarrow X_1$; $R_4 \leftarrow Y_1$;
$i \leftarrow m-2$; $s \leftarrow 1$; $k \leftarrow 0$;
Tant que $i \geq 0$, répéter $\Gamma(k, s)$:
$\gamma(k)$
$s \leftarrow k - D_i + 1$
$k \leftarrow (k+1) \times (/s)$;
$i \leftarrow i - s$;
Retourner (R1, R2)

Algorithme 8
(algorithme sur courbe elliptique protégé)

REVENDICATIONS

1. Procédé de calcul cryptographique caractérisé en ce que, pour exécuter un bloc d'instructions choisi (Π_j) en fonction d'une variable d'entrée (D_i) parmi N blocs d'instructions prédéfinis (Π_1, ..., Π_N), on exécute un nombre prédéfini (L_j) de fois un bloc commun ($\Gamma(k,s)$) aux N blocs d'instructions prédéfinis (Π_1, ..., Π_N), le nombre prédéfini (L_j) étant associé au bloc d'instructions choisi (Π_j).

2. Procédé selon la revendication 1, dans lequel le nombre prédéfini (L_j) est variable d'un bloc d'instructions prédéfini (Π_1, ..., Π_N) à l'autre.

3. Procédé selon l'une des revendications 1 à 2, dans lequel le bloc commun ($\Gamma(k,s)$) comprend au moins une instruction (γk) de calcul équivalente vis-à-vis d'une attaque à canal caché à une instruction de calcul de chaque bloc prédéfini (Π_1, ..., Π_N).

4. Procédé selon la revendication 3, dans lequel le bloc commun ($\Gamma(k,s)$) comprend également une instruction de mise à jour d'un pointeur de boucle (k) indiquant un nombre d'exécutions déjà exécutées du bloc élémentaire commun ($\Gamma(k,s)$).

5. Procédé selon la revendication 3 ou la revendication 4, dans lequel le bloc commun ($\Gamma(k,s)$) comprend également une instruction de mise à jour d'un pointeur d'état (s) indiquant si le nombre prédéfini (L_j) a été atteint.

6. Procédé selon la revendication 4 ou la revendication 5, dans lequel la valeur du pointeur de

boucle (k) et / ou la valeur du pointeur d'état (s) sont fonction de la valeur de la variable d'entrée (D_i) et / ou du nombre d'instructions du bloc d'instructions (P_j) associé à la valeur de donnée d'entrée.

7. Procédé selon l'une des revendications 1 à 6, dans lequel, pour réaliser successivement plusieurs blocs d'instructions choisis parmi les N blocs d'instructions prédéfinis (Π_1, ..., Π_N), chaque bloc d'instruction choisi (Πj) étant sélectionné en fonction d'une variable d'entrée (D_i) associée à un indice d'entrée (i),

on exécute un nombre total (L_T) de fois le bloc élémentaire commun ($\Gamma(k,s)$), le nombre total (L_T) étant égal à une somme des nombres prédéfinis (L_j) associés à chaque bloc d'instructions choisi (Π_j).

8. Procédé selon la revendication 7, au cours duquel un même bloc d'instructions peut être choisi plusieurs fois selon la variable d'entrée associée à l'indice d'entrée (i).

9. Procédé selon l'une des revendications 7 ou 8, dans lequel la valeur du pointeur de boucle (k) et / ou la valeur du pointeur d'état (s) et / ou la valeur de la variable d'entrée (D_i) et / ou le nombre d'instructions du bloc d'instructions (Π_j) associé à la valeur de la donnée d'entrée (D_i) sont liées par une ou des fonctions mathématiques.

10. Procédé selon la revendication 9, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant les étapes suivantes :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $i \leftarrow M-1$

Tant que $i \geq 0$, répéter le bloc commun $\Gamma(k, s)$:

$k \leftarrow (/s) \times (k+1) + s \times 2 \times (/D_i)$

$s \leftarrow (k \bmod 2) + (k \operatorname{div} 2)$

$\gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{k \bmod 2}$

$i \leftarrow i - s$

Retourner R_0.

11. Procédé selon la revendication 9, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant les étapes suivantes :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $i \leftarrow M-1$; $k \leftarrow 1$

Tant que $i \geq 0$, répéter le bloc commun $\Gamma'(k,s)$:

$k \leftarrow (D_i)$ ET $(/k)$

$\gamma'(s, k)$: $R_0 \leftarrow R_0 \times R_k$

$i \leftarrow i - (/k)$

Retourner R_0.

12. Procédé selon la revendication 9, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant les étapes suivantes :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $i \leftarrow 0$; $k \leftarrow 1$

Tant que $i \leq M-1$, répéter le bloc $\Gamma(k, s)$:

$k \leftarrow k \oplus D_i$

$\gamma(k)$: $R_k \leftarrow R_k \times R_1$

$i \leftarrow i+k$

Retourner R_0.

13. Procédé selon la revendication 9, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit

(D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant les étapes suivantes :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $R_2 \leftarrow A^3$;

$D_{-1} \leftarrow 0$; $i \leftarrow M-1$; $s \leftarrow 1$

Tant que $i \geq 0$, répéter le bloc $\Gamma(k, s)$:

$k \leftarrow (/s) \times (k+1) + s \times (D_i + 2 \times (D_i \text{ ET } D_{i-1}))$

$s \leftarrow /((k \bmod 2) \oplus (k \text{ div } 4))$

$\gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{s \times (k \text{ div } 2)}$

$i \leftarrow i - s \times (k \bmod 2 + 1)$

Retourner R_0.

14. Procédé selon la revendication 9, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant les étapes suivantes :

Initialisation :

$R_0 \leftarrow 1$; $R_1 \leftarrow A$; $R_2 \leftarrow A^3$;

$D_{-1} \leftarrow 0$; $i \leftarrow M-1$; $s \leftarrow 1$

Tant que $i \geq 0$, répéter :

$k \leftarrow (/s) \times (k+1)$

$s \leftarrow s \oplus D_i \oplus ((D_{i-1} \text{ ET } (k \bmod 2))$

$\Gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{k \times s}$

$i \leftarrow i - k \times s - (/D_i)$

Retourner R_0.

15. Procédé selon l'une des revendications 7 ou 8, dans lequel les liens entre la valeur du pointeur de boucle (k) et / ou la valeur du pointeur d'état (s) et / ou la valeur de la variable d'entrée (D_i) et / ou le nombre d'instructions du bloc d'instructions (Π_j) associé à la valeur de la donnée d'entrée (D_i) sont définis par une table à plusieurs entrées telle qu'une matrice (U(k,l)).

16. Procédé selon la revendication 15, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$, D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant l'étape suivante :

Tant que $i \geq 0$, répéter le bloc $\Gamma(k, s)$:

$k \leftarrow (/s) \times (k+1) + s \times 2 \times (/D_i)$;

$s \leftarrow U(k,1)$

$\gamma(k, s)$: $R_0 \leftarrow R_0 \times R_{U(k,0)}$

$i \leftarrow i - s$

où (U(k,1)) est la matrice suivante :

$$(U(k,1))_{\substack{0 \leq k \leq 2 \\ 0 \leq 1 \leq 1}} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$$

17. Procédé selon la revendication 15, utilisé dans la mise en œuvre d'un calcul d'exponentiation de type $B = A^D$ selon l'algorithme (M, M^3), D étant un nombre entier de M bits, chaque bit (D_i) de D correspondant à une variable d'entrée d'indice d'entrée i, le procédé comprenant l'étape suivante :

Tant que $i \geq 0$, répéter le bloc commun $\Gamma(k, s)$:

$k \leftarrow (/s) \times (k+1) + s \times (D_i + 2 \times (D_i \text{ ET } D_{i-1}))$

$s \leftarrow U(k,2)$

$\gamma(k,s)$: $R_0 \leftarrow R_0 \times R_{U(k,0)}$;

$i \leftarrow i - U(k,1)$

où (U(k,1)) est la matrice suivante :

$$(U(k,1))_{\substack{0 \leq k \leq 5 \\ 0 \leq 1 \leq 2}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 2 & 1 \end{pmatrix}$$

18. Procédé selon la revendication 15, utilisé dans la mise en œuvre d'un calcul sur une courbe elliptique en coordonnées affines, calcul utilisant des opérations de type addition ou doublement de points, et dans lequel on réalise l'étape suivante :

Tant que $i \geq 0$, répéter $\Gamma(k, s)$:

$\gamma(k)$: $R_{U(k,0)} \leftarrow R_1 + R_3$;

$R_{U(k,1)} \leftarrow R_{U(k,1)} + R_{U(k,2)}$;

$R_5 \leftarrow R_2 / R_1$; $R_{U(k,3)} \leftarrow R_1 + R_5$;

$R_{U(k,4)} \leftarrow R_5^2$;

$R_{U(k,4)} \leftarrow R_{U(k,4)} + a$;

$R_1 \leftarrow R_1 + R_{U(k,5)}$;

$R_2 \leftarrow R_1 + R_{U(k,6)}$; $R_6 \leftarrow R_1 + R_{U(k,7)}$;

$R_5 \leftarrow R_5 . R6$; $R_2 \leftarrow R_2 + R_5$

$s \leftarrow k - D_i + 1$

$k \leftarrow (k+1) \times (/s)$;

$i \leftarrow i - s$;

où (U(k,l)) est la matrice suivante :

$$(U(k,l))_{\substack{0\leq k\leq 1\\ 0\leq l\leq 10}} = \begin{pmatrix} 1 & 2 & 4 & 1 & 6 & 6 & 4 & 3 \\ 6 & 6 & 3 & 5 & 1 & 5 & 2 & 6 \end{pmatrix}$$

19. Procédé d'obtention d'un bloc élémentaire commun ($\Gamma(k,s)$) à N blocs d'instructions (Π_1, ..., Π_N) prédéfinis, procédé susceptible d'être utilisé pour la mise en œuvre d'un procédé de calcul cryptographique selon l'une des revendications 1 à 12, le procédé étant caractérisé en ce qu'il comprend les étapes suivantes :

E1 : décomposition de chaque bloc d'instructions prédéfini (Π_1, ..., Π_N) en une suite de blocs élémentaires (γ) équivalents vis-à-vis d'une attaque à canal caché, et classement de l'ensemble des blocs élémentaires,

E2 : recherche d'un bloc élémentaire commun ($\gamma(k,s)$) équivalents à tous les blocs élémentaires (γ) de tous les blocs d'instructions prédéfinis,

E3 : recherche d'un bloc commun ($\Gamma(k,s)$) comprenant au moins le bloc élémentaire commun ($\gamma(k,s)$ précédemment obtenu et une instruction de mise à jour d'un pointeur de boucle (k) tel que une exécution du bloc élémentaire commun associée à la valeur du pointeur de boucle (k) et une exécution du bloc élémentaire de rang égal à la valeur du pointeur de boucle (k) soient identiques.

20. Procédé selon la revendication 19, caractérisé en ce que au cours de l'étape E1, on ajoute au moins une instruction fictive à au moins un bloc d'instructions prédéfini.

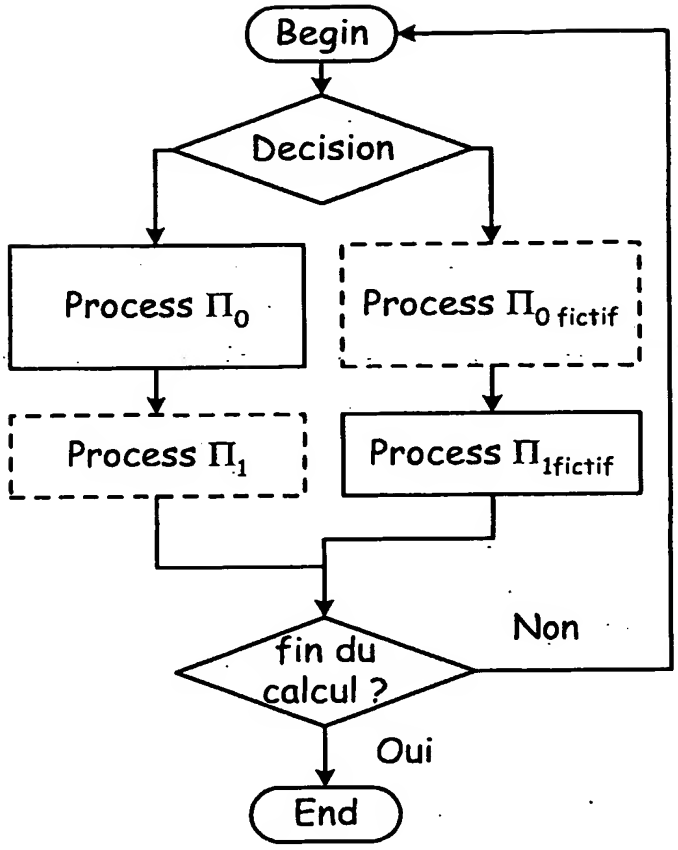


Fig. 1

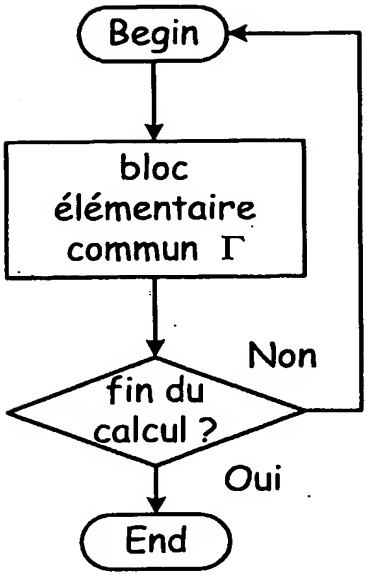


Fig. 2

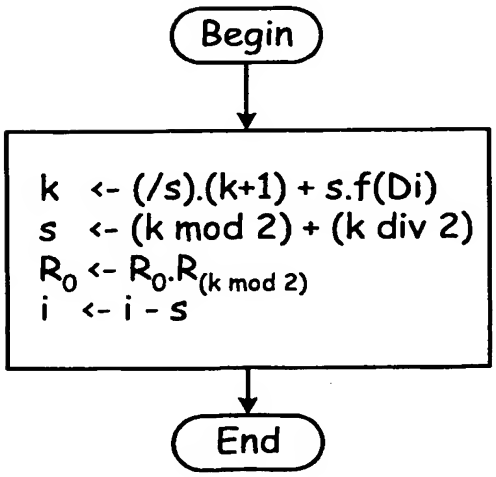


Fig. 3

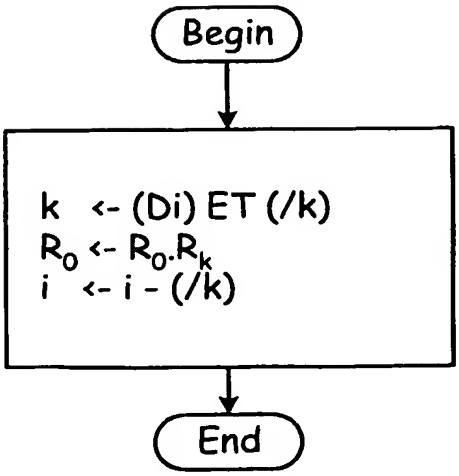


Fig. 4